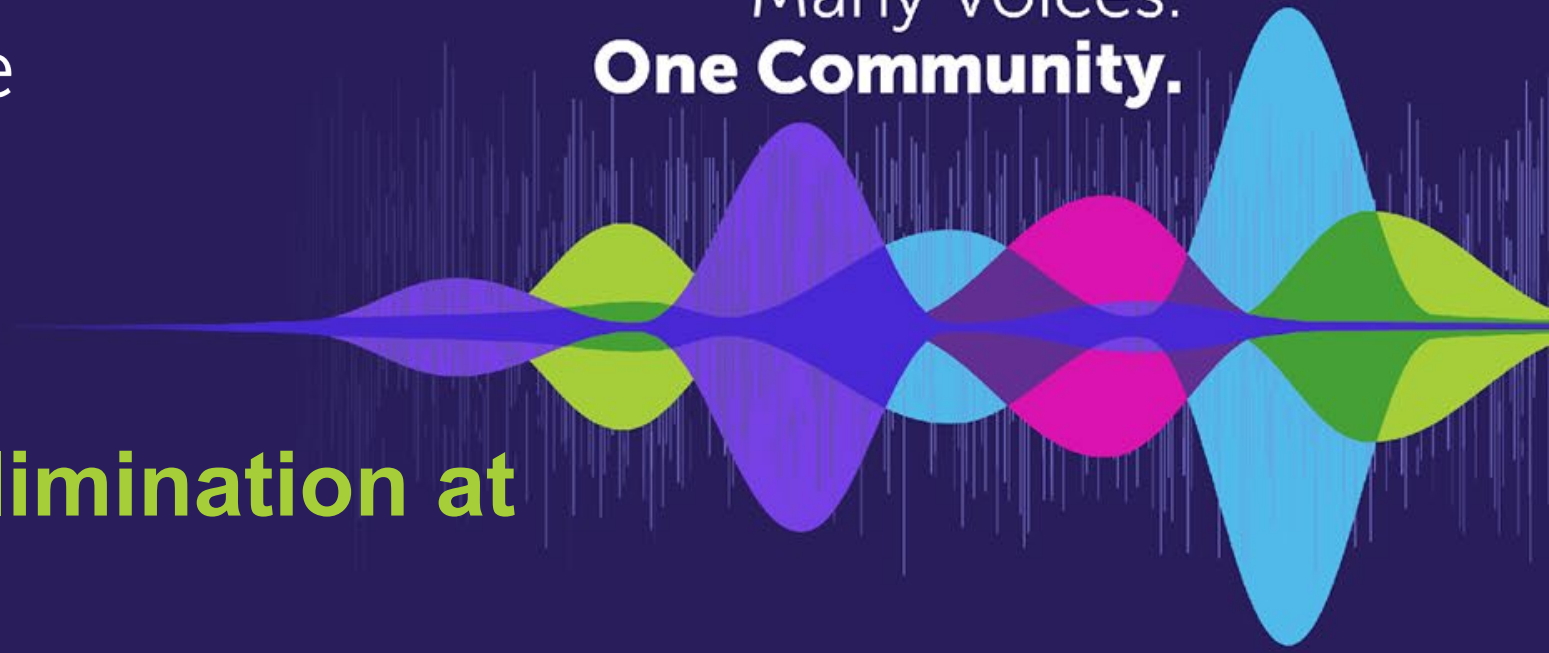
RSAC 2025 Conference

Many Voices. One Community.



SESSION ID: CRYP-W01A

Masking Gaussian Elimination at Arbitrary Order

with Application to Multivariate- and Code-Based PQC

Quinten Norga

PhD Candidate
COSIC, KU Leuven

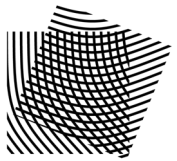
Co-Authors: Suparna Kundu, Uttam Kumar Ojha, Anindya Ganguly, Angshuman Karmakar, Ingrid Verbauwhede

Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the presenters individually and, unless expressly stated to the contrary, are not the opinion or position of RSA Conference LLC or any other co-sponsors. RSA Conference LLC does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented.

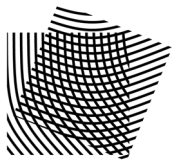
Attendees should note that sessions may be audio- or video-recorded and may be published in various media, including print, audio and video formats without further notice. The presentation template and any media capture are subject to copyright protection.

© 2025 RSA Conference LLC or its affiliates. The RSAC and RSAC CONFERENCE logos and other trademarks are proprietary. All rights reserved.

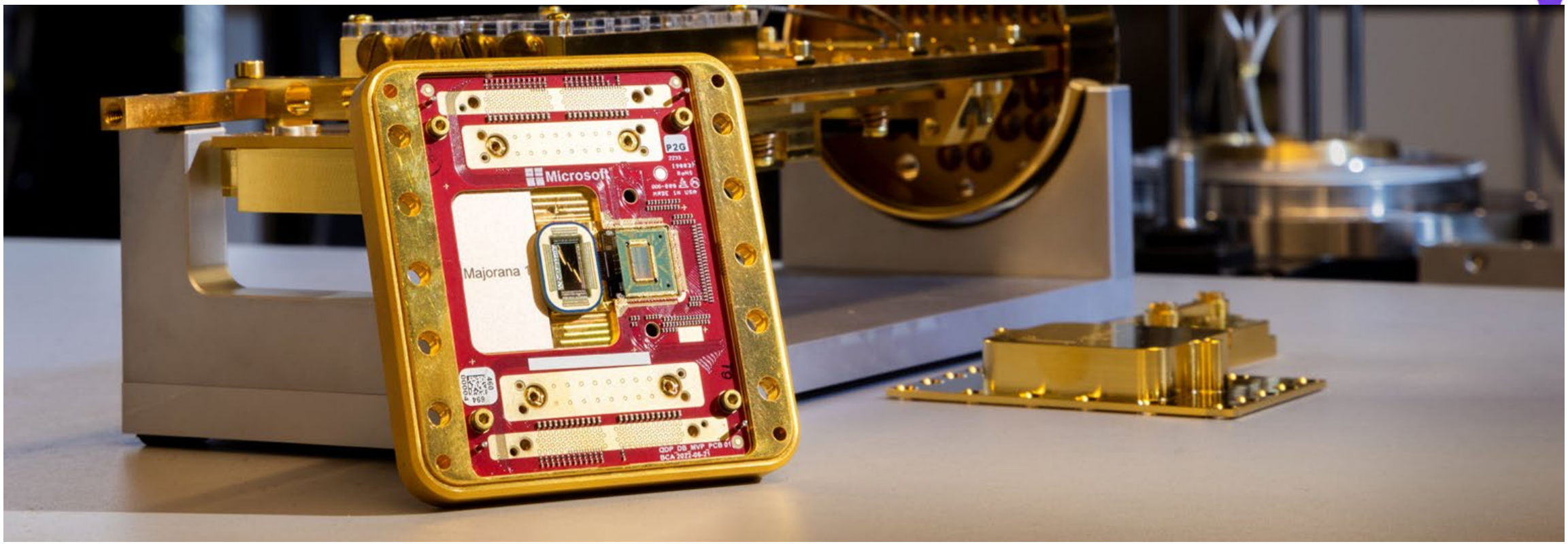


Outline

- Introduction: Multivariate- and Code-Based PQC
- Masking Gaussian Elimination w/ Back Substitution
- Implementation & Evaluation on Arm Cortex-M4



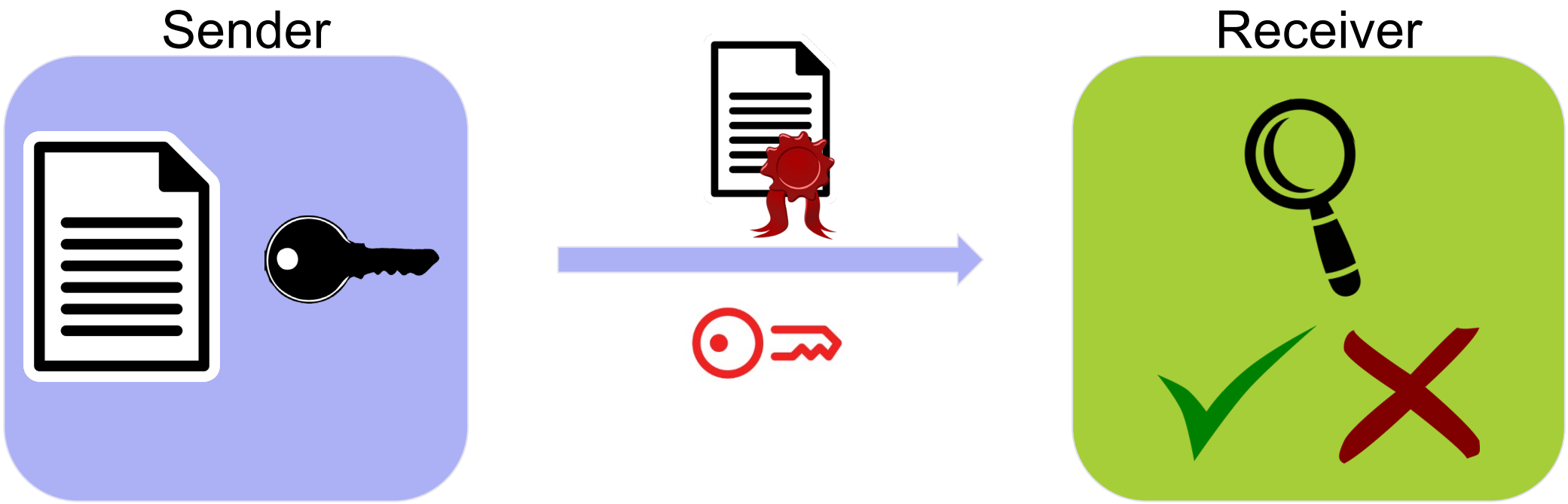
Post-Quantum Cryptography (PQC)



Majorana 1

[Microsoft]

Digital Signature Algorithm (DSA)



COSIC

PQC DSA Standardization

NIST

FIPS 204

Federal Information Processing Standards Publication

Module-Lattice-Based Digital Signature Standard

Category: Computer Security Subcategory: Cryptography

[FIPS 204]

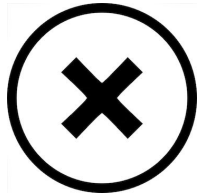
FIPS 205

Federal Information Processing Standards Publication

Stateless Hash-Based Digital Signature Standard

Category: Computer Security Subcategory: Cryptography

[FIPS 205]



and many more…

[NIST]

COSIC

Multivariate- and Code-Based DSA

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

System of Linear Equations

COSIC

Linear Algebra 101: Gaussian Elimination

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} & b_1 \\ a_{21} & a_{22} & \dots & a_{2m} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mm} & b_m \end{bmatrix}$$

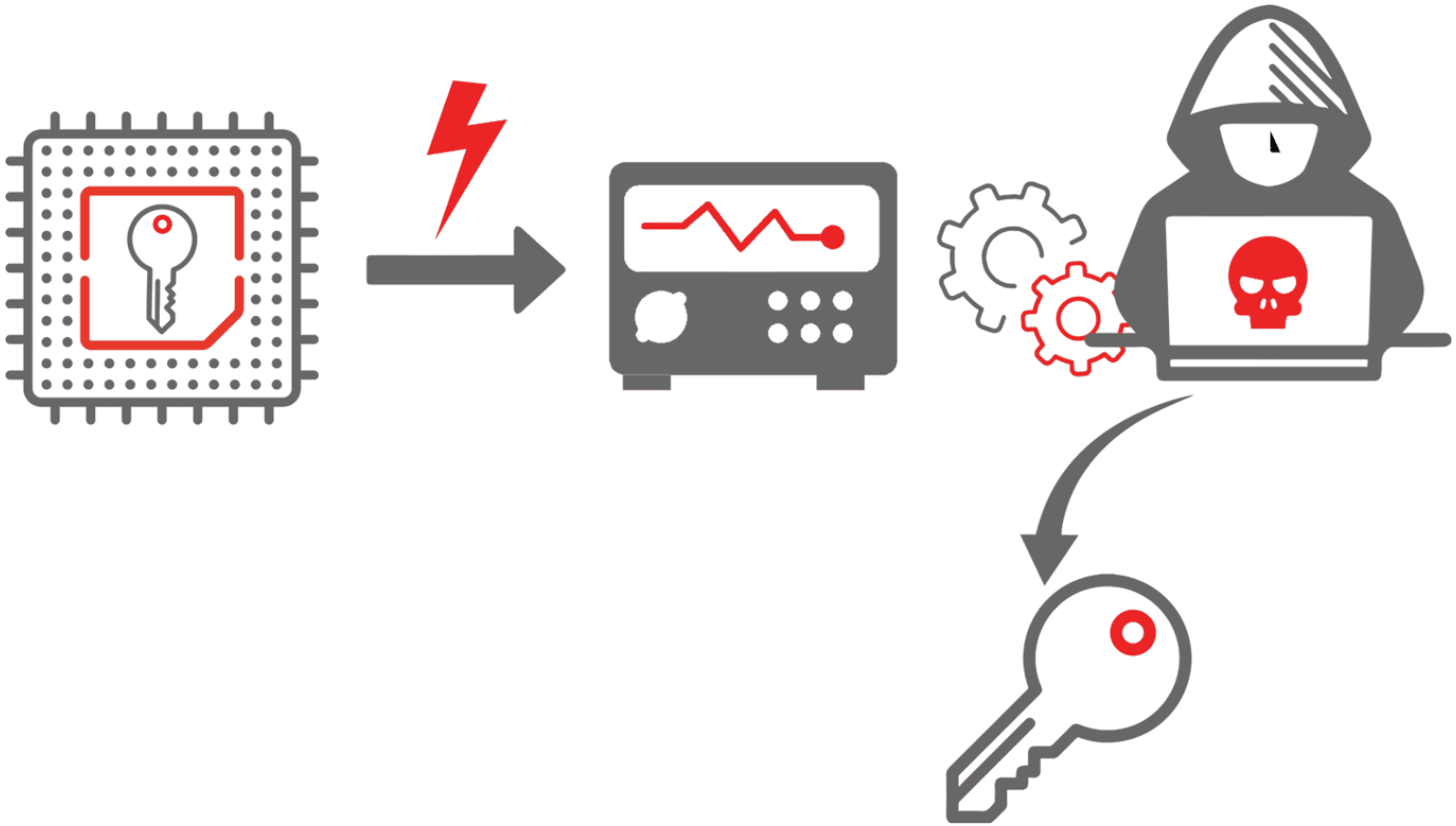
COSIC

Row-Echelon Form

$$\begin{bmatrix} 1 & a'_{12} & \dots & a'_{1m} & b'_1 \\ 0 & 1 & \dots & a'_{2m} & b'_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & b'_m \end{bmatrix}$$

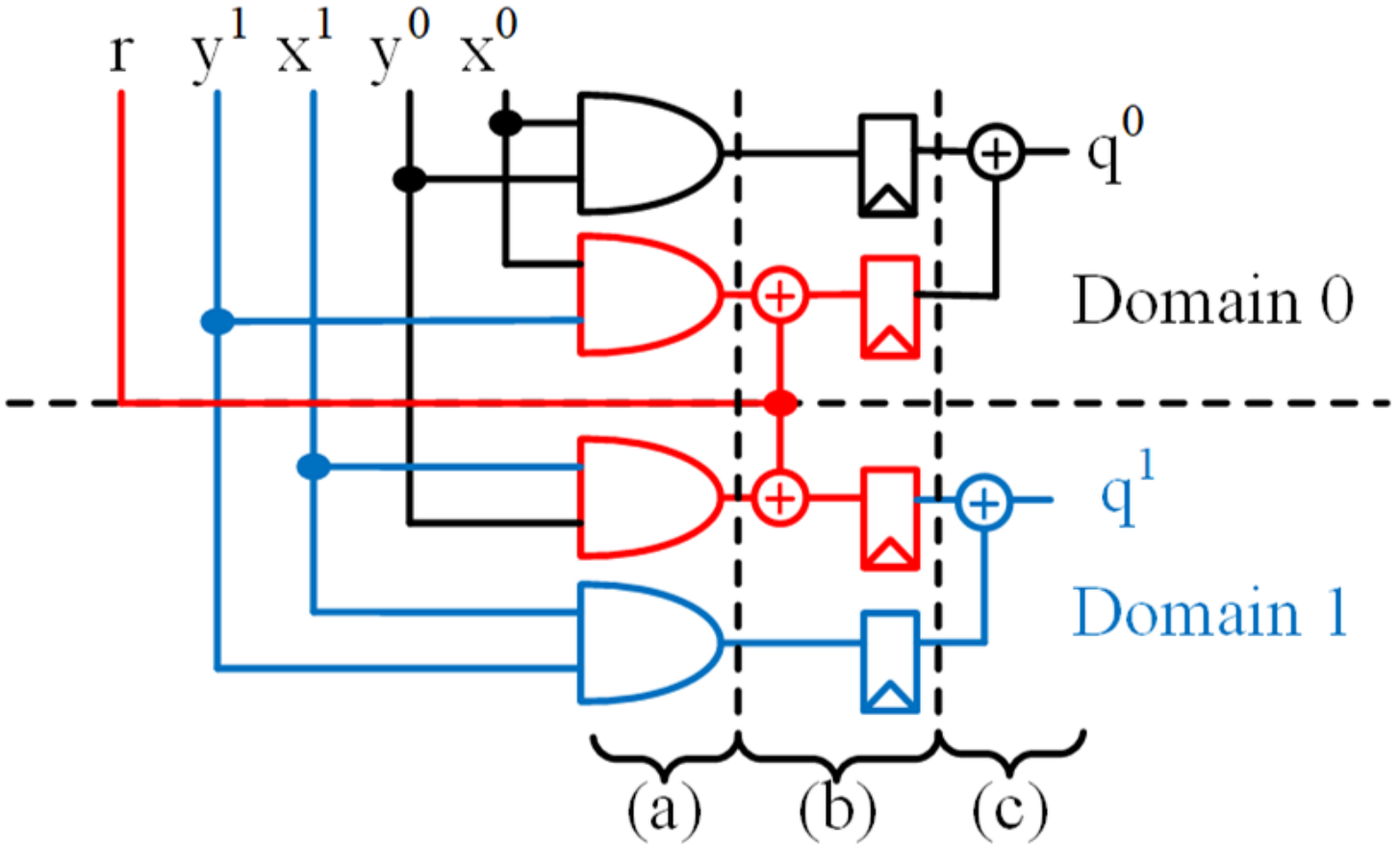
Row-Operations: Swap, Multiply, Add

Side-Channel Attacks (SCAs)



[Secure-IC]

Masking



[Gross et al.]

COSIC

Masked System of Linear Equations

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

$$a^1_{xy} * a^2_{xy} = a_{xy}$$

Multiplicative Sharing

COSIC

RSAC 2025 Conference

Masking Gaussian Elimination

Step 1: Make Pivot Non-Zero

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

$$a^1_{11} + a^2_{11} = a_{11} \stackrel{?}{=} 0$$

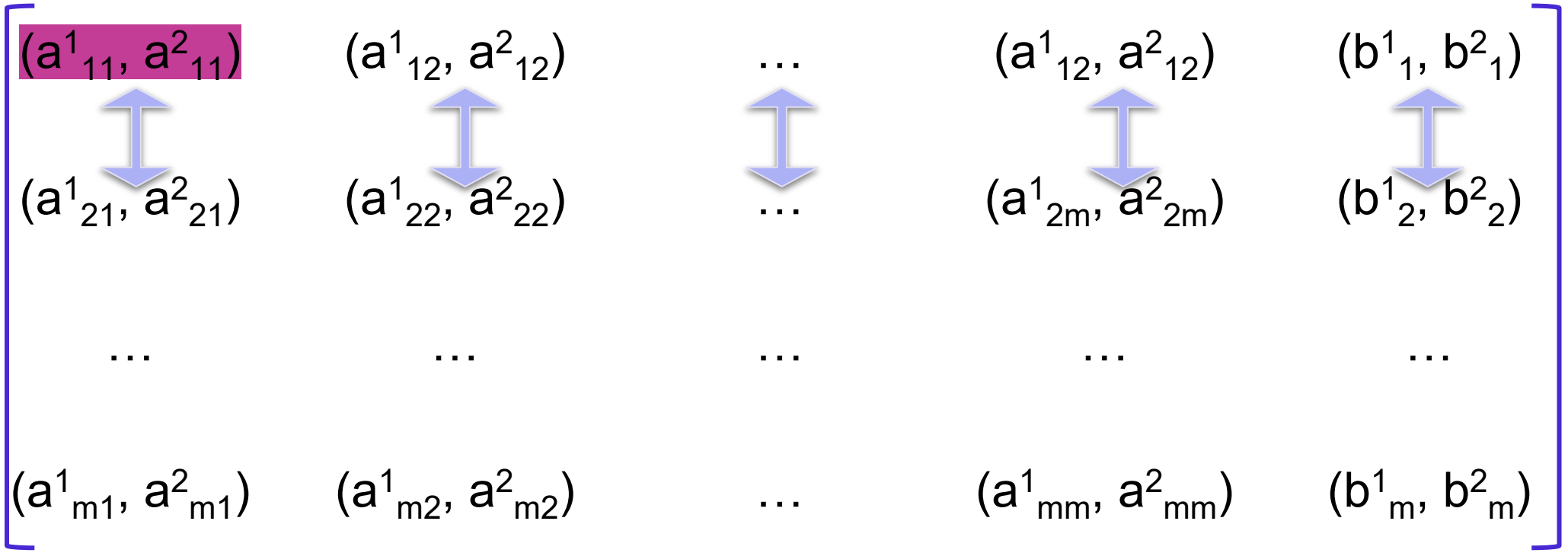
COSIC

SecNonzero [CC24]

$$a^1_{11} + a^2_{11} = a_{11} \stackrel{?}{=} 0$$

- Returns masked bool (1/0)
- Unmasking *leaks* information

Step 1: Make Pivot Non-Zero



$a^1_{11} + a^2_{11} = a_{11} \neq 0$

Not Constant Time!

SecCondAdd

- Conditional Addition
  - Add rows to Pivot row…
  - or add 0 to Pivot row

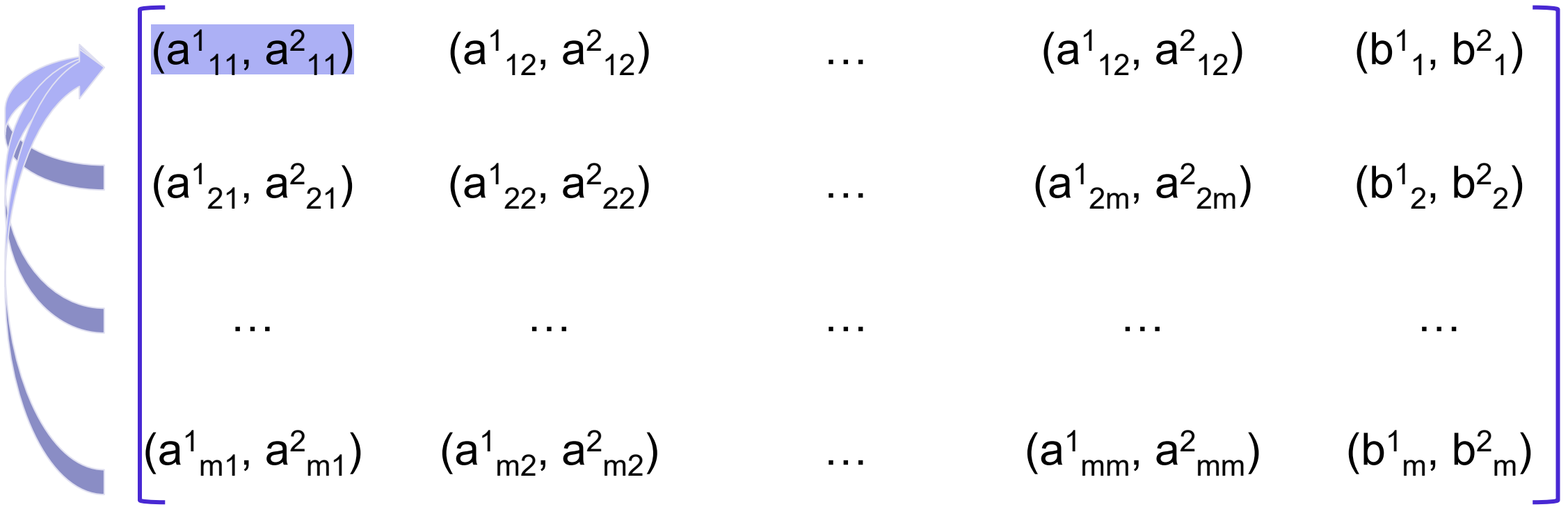
$$a^1_{11} + a^2_{11} = a_{11} = 0$$

$$a^1_{11} + a^2_{11} = a_{11} \neq 0$$

  - Details in paper

COSIC

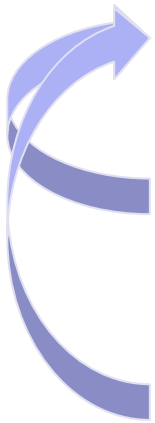
Step 1: Make Pivot Non-Zero



SecCondAdd (CT)

COSIC

Step 1: Make Pivot Non-Zero



$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

Early Stop (details in paper)

COSIC

Step 2: Check Pivot

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

SecNonzero

SecNonzero [CC24] + Unmask

$$a^1_{11} + a^2_{11} = a_{11} \stackrel{?}{=} 0$$

- Leak if pivot is (non-)zero
- No unique solution → OK!

COSIC

Step 2: Check Pivot

$$\begin{bmatrix} a_{11}=0 & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

Start over!

Step 3: Make Pivot 1

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

$$a_{11} * a_{11}^{-1} = 1$$

COSIC

B2Minv

- Multiplication prefers multiplicative masking
  - B2M conversion [GenProQui11]

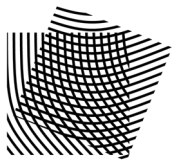
$$(a^1_{11} + a^2_{11}) = (a'^1_{11} * a'^2_{11})$$

SecScalarMult

- Strictly* linear operations

$$(a^1_{11} + a^2_{11}) * a'^1_{11} * a'^2_{11}$$

- Total Cost:
  - 1 B2M conversion
  - NO M2B conversion



COSIC

Step 3: Make Pivot 1

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

$$(a^1_{11} + a^2_{11}) = 1$$

COSIC

Step 3: Make Pivot 1

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

SecScalarMult on Row 1

Step 4: Make Lower Elements 0

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

Subtract first-row from rows below: SecMultSub

COSIC

Step 4: Make Lower Elements 0

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

Subtract first-row from rows below: SecMultSub

New Pivot: Repeat Step 1-4

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

$$a^1_{22} + a^2_{22} = a_{22} \stackrel{?}{=} 0$$

Resulting Matrix

$$\begin{bmatrix} (a^1_{11}, a^2_{11}) & (a^1_{12}, a^2_{12}) & \dots & (a^1_{12}, a^2_{12}) & (b^1_1, b^2_1) \\ (a^1_{21}, a^2_{21}) & (a^1_{22}, a^2_{22}) & \dots & (a^1_{2m}, a^2_{2m}) & (b^1_2, b^2_2) \\ \dots & \dots & \dots & \dots & \dots \\ (a^1_{m1}, a^2_{m1}) & (a^1_{m2}, a^2_{m2}) & \dots & (a^1_{mm}, a^2_{mm}) & (b^1_m, b^2_m) \end{bmatrix}$$

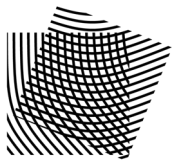
Final step: back-substitution

COSIC

SecBackSub

- See paper for details

- Reduce cost through early unmasking



Application to PQC DS Schemes

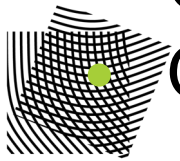
- Cost determined by
  - **m** (matrix dimension)
  - **q** (modulus)
  - **n** (# shares)

- NIST Additional DS Round 2 & Korean PQC Candidates
  - UOV, MAYO, SNOVA, QR-UOV, MQ-Sign

UOV Implementation on Arm Cortex-M4

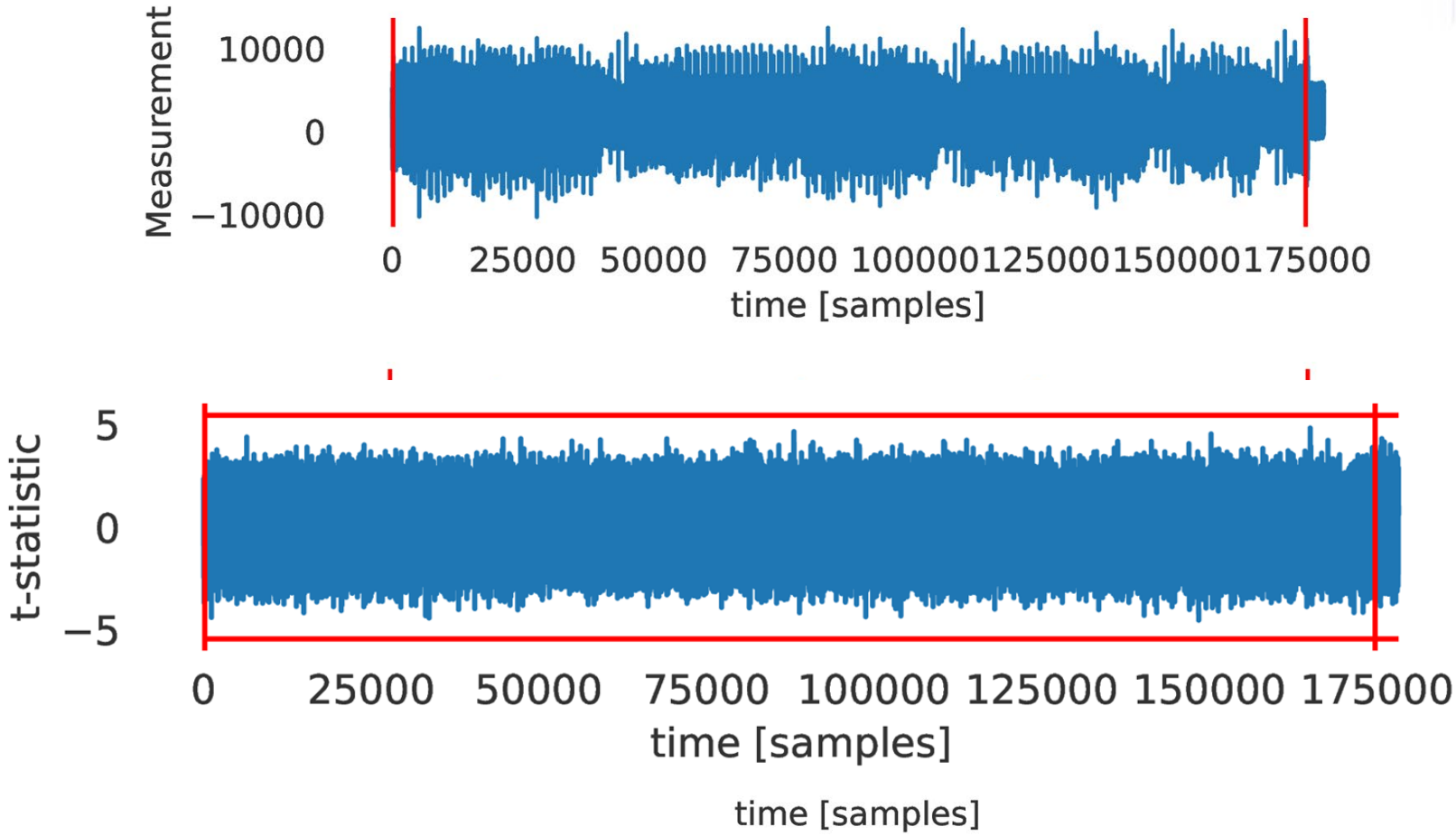
Scheme	Operation	Umasked	Masked		
			1st-order	2nd-order	3rd-order
UOV-I (m=44 q=256)	GE (Total)	1034	**15649 (15.1×)**	**39587 (38.3×)**	**69186 (66.9×)**
	SecRowEch	999	15575 (15.6×)	39469 (39.5×)	69021 (69.1×)
	Step 1	80	1998 (25×)	5642 (70.5×)	9901 (123.8×)
	Step 2		39	108	193
	Step 3		598	1335	2426
	Step 4		12938	32383	56498
	SecBackSub	32	74 (2.3×)	118 (3.7×)	164 (5.1×)
UOV-III	GE (Total)	4117	**62680 (15.2×)**	**157901 (38.4×)**	**275768 (67×)**
UOV-V	GE (Total)	9336	**143361 (15.4×)**	**360590 (38.6×)**	**1025942 (109.9×)**

- STM32 Discovery
- C code (arm-none-eabi-gcc)
- On-the-fly TRNG



COSIC

Security Evaluation in COSIC Lab



Countermeasure OFF

Countermeasure ON

Test Vector Leakage Assessment (TVLA)

Mitigation of micro-architectural leakages

COSIC

RSAC 2025 Conference

Key Takeaways

How to Apply our results?

Conclusion

- **Real-World** requires **Secure & Efficient** implementations
  - Design of PQC schemes
  - Implementation of PQC schemes

- PQC transition
  - Crypto inventory
  - Agility: many standards!

COSIC

RSAC 2025 Conference

Many Voices.
One Community.

Thank You!

Questions?

RSAC 2025 Conference

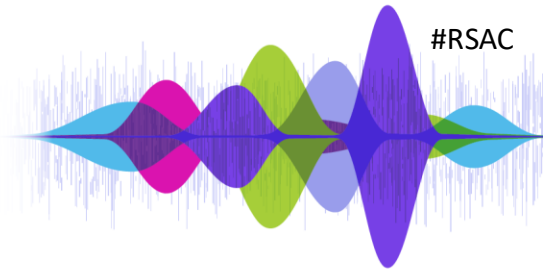
Many Voices.
One Community.

SESSION ID: CRYP-W01B

Hardware and Software Implementations: Post-Quantum Online/Offline Signatures

James Howe

Staff Research Scientist
SandboxAQ
https://www.linkedin.com/in/jameshowe1729/



Outline

- Introduction
  - A refresher on Post-Quantum Cryptography
  - What is the Online/Offline paradigm
  - Why could this intersection work?
- Our Contributions
- Results

RSAC 2025 Conference

Introduction

Why post-quantum cryptography matters

Many Voices.
One Community.

Why Post-Quantum Matters

The Quantum Threat

Practical quantum computers are coming

RSA & ECC vulnerable via Shor's algorithm

NIST's PQC Efforts

Standardizing quantum-secure cryptosystems.

FIPS standards: ML-KEM, ML-DSA (Dilithium), SLH-DSA, & Falcon.

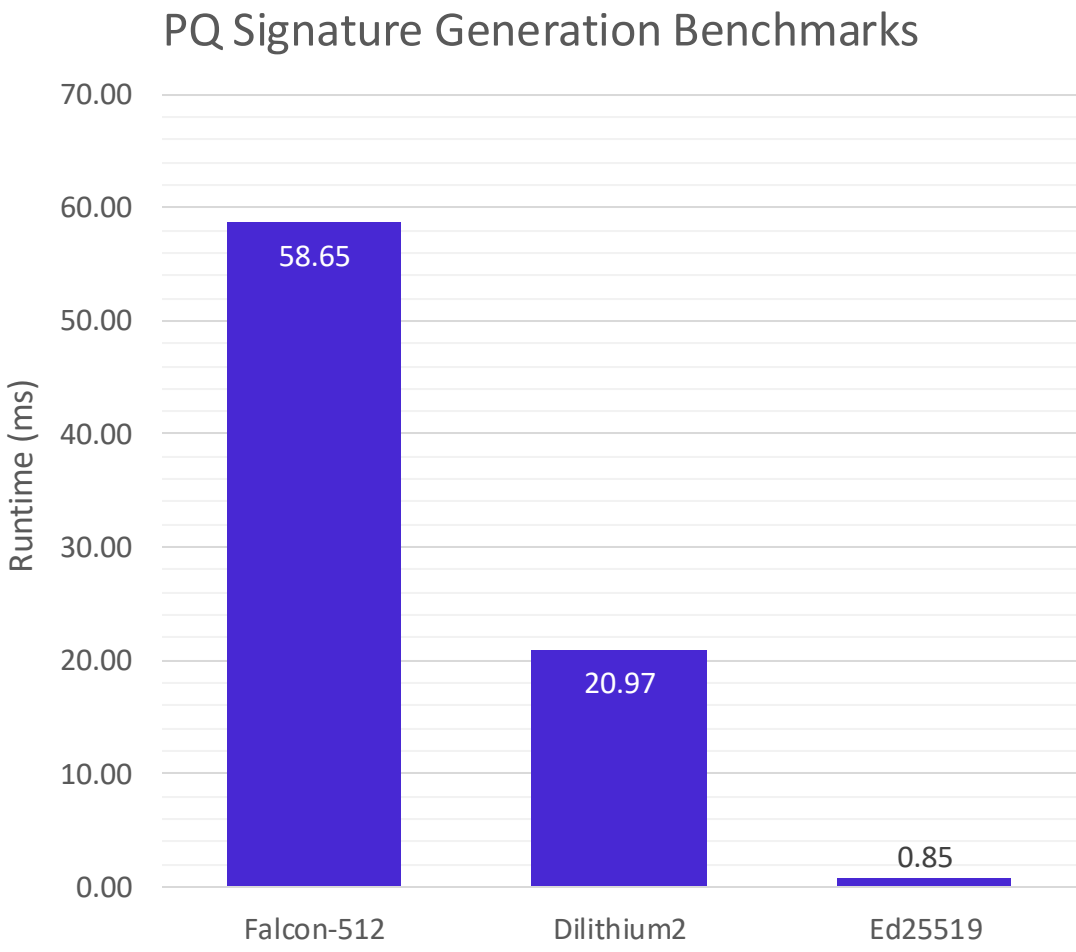
PQC Signature Issues

High computational costs → Slower than classical signatures.

Impacts smart cards, IoT, embedded devices.

PQ Signature Embedded Performance

- Why This Matters
  - Ed25519 is fast (used today in SSH, TLS, etc.).
  - Falcon & Dilithium are much slower due to heavy math operations.
- **Impact**: Real-time applications struggle with PQC signatures.
- **Solution**: Online/Offline signatures
  - Precompute heavy operations “offline” to make signing faster.



RSAC 2025 Conference

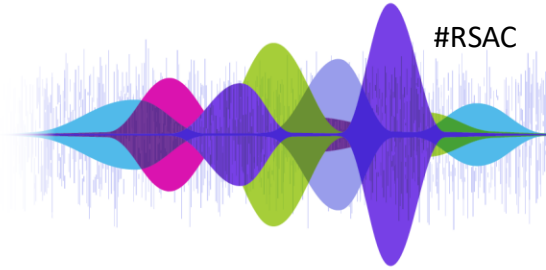
Introduction

What are online/offline signatures?

Many Voices.
One Community.

What are online/offline signatures?

- Introduced by Even, Goldreich, & Micali (1989, 1996)
- Allows efficient signing on devices with limited computational power.
  - 💳 Smart Cards – auth for ID cards, secure access, etc.
  - 🏪 POS (Point-of-Sale) Payment Systems – NFC & RFID-based contactless payments.
  - 🔐 Secure Transactions – Used in low-power devices where fast signing is required.



On-Line/Off-Line Digital Signatures*

Shimon Even
Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel
even@cs.technion.ac.il

Oded Goldreich
Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel
oded@wisdom.weizmann.ac.il

Silvio Micali
Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, U.S.A.
silvio@theory.lcs.mit.edu

Communicated by Gilles Brassard

Received 19 August 1992 and revised 21 December 1994

Abstract. A new type of signature scheme is proposed. It consists of two phases. The first phase is performed off-line, before the message to be signed is even known. The second phase is performed on-line, once the message to be signed is known, and is supposed to be very fast. A method for constructing such on-line/off-line signature schemes is presented. The method uses one-time signature schemes, which are very fast, for the on-line signing. An ordinary signature scheme is used for the off-line stage.

In a practical implementation of our scheme, we use a variant of Rabin's signature scheme (based on factoring) and DES. In the on-line phase all we use is a moderate amount of DES computation and a single modular multiplication. We stress that the costly modular exponentiation operation is performed off-line. This implementation is ideally suited for electronic wallets or smart cards.

Key words. Digital signatures, Integer factorization, RSA, DES, One-time signature schemes, Error-correcting codes, Chosen message attack.

What are online/offline signatures?

- Splits signing into two phases:
  - Offline Phase 🛠️ → Precompute expensive operations before signing (e.g., modular exponentiation)
  - Online Phase ⚡ → Fast signing when needed using precomputed data

On-Line/Off-Line Digital Signatures*

Shimon Even
Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel
even@cs.technion.ac.il

Oded Goldreich
Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel
oded@wisdom.weizmann.ac.il

Silvio Micali
Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, U.S.A.
silvio@theory.lcs.mit.edu

Communicated by Gilles Brassard

Received 19 August 1992 and revised 21 December 1994

Abstract. A new type of signature scheme is proposed. It consists of two phases. The first phase is performed off-line, before the message to be signed is even known. The second phase is performed on-line, once the message to be signed is known, and is supposed to be very fast. A method for constructing such on-line/off-line signature schemes is presented. The method uses one-time signature schemes, which are very fast, for the on-line signing. An ordinary signature scheme is used for the off-line stage.

In a practical implementation of our scheme, we use a variant of Rabin's signature scheme (based on factoring) and DES. In the on-line phase all we use is a moderate amount of DES computation and a single modular multiplication. We stress that the costly modular exponentiation operation is performed off-line. This implementation is ideally suited for electronic wallets or smart cards.

Key words. Digital signatures, Integer factorization, RSA, DES, One-time signature schemes, Error-correcting codes, Chosen message attack.

What are online/offline signatures?

- Offline precomputation phase 🛠️
  - Using a regular signature scheme, generates sign/verify long-term keys
  - Generates one-time sign/verify keys
  - Signs the one-time verify key with long term key
- Online Phase ⚡
  - Quickly verifies the one-time key
  - Signs the message/challenge with the one-time key
  - Quickly verifies the one-time signature created
- ⚠️ We are verifying twice, only works if 2*verifications are much faster than one regular signing

On-Line/Off-Line Digital Signatures*

Shimon Even
Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel
even@cs.technion.ac.il

Oded Goldreich
Department of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel
oded@wisdom.weizmann.ac.il

Silvio Micali
Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, U.S.A.
silvio@theory.lcs.mit.edu

Communicated by Gilles Brassard

Received 19 August 1992 and revised 21 December 1994

Abstract. A new type of signature scheme is proposed. It consists of two phases. The first phase is performed off-line, before the message to be signed is even known. The second phase is performed on-line, once the message to be signed is known, and is supposed to be very fast. A method for constructing such on-line/off-line signature schemes is presented. The method uses one-time signature schemes, which are very fast, for the on-line signing. An ordinary signature scheme is used for the off-line stage.

In a practical implementation of our scheme, we use a variant of Rabin's signature scheme (based on factoring) and DES. In the on-line phase all we use is a moderate amount of DES computation and a single modular multiplication. We stress that the costly modular exponentiation operation is performed off-line. This implementation is ideally suited for electronic wallets or smart cards.

Key words. Digital signatures, Integer factorization, RSA, DES, One-time signature schemes, Error-correcting codes, Chosen message attack.

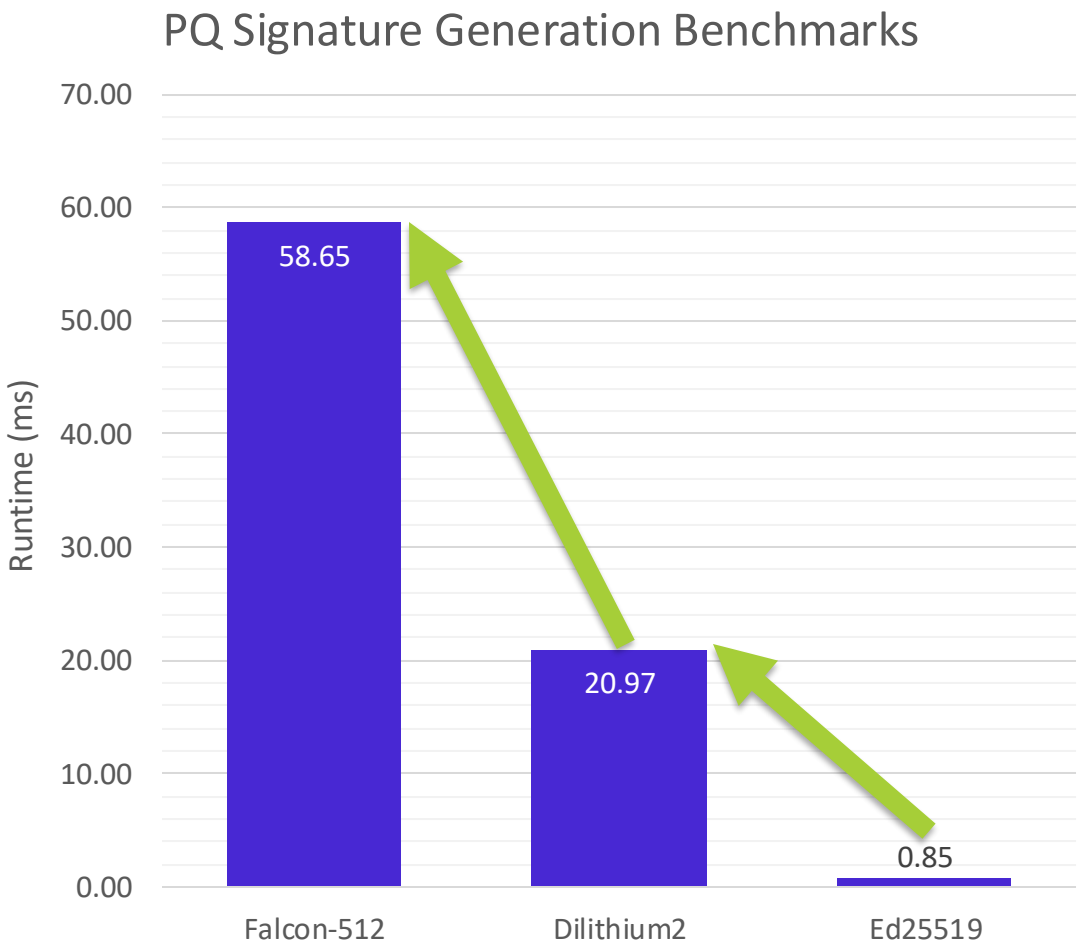
RSAC 2025 Conference

Post-Quantum Online/Offline Signatures

Why Falcon?

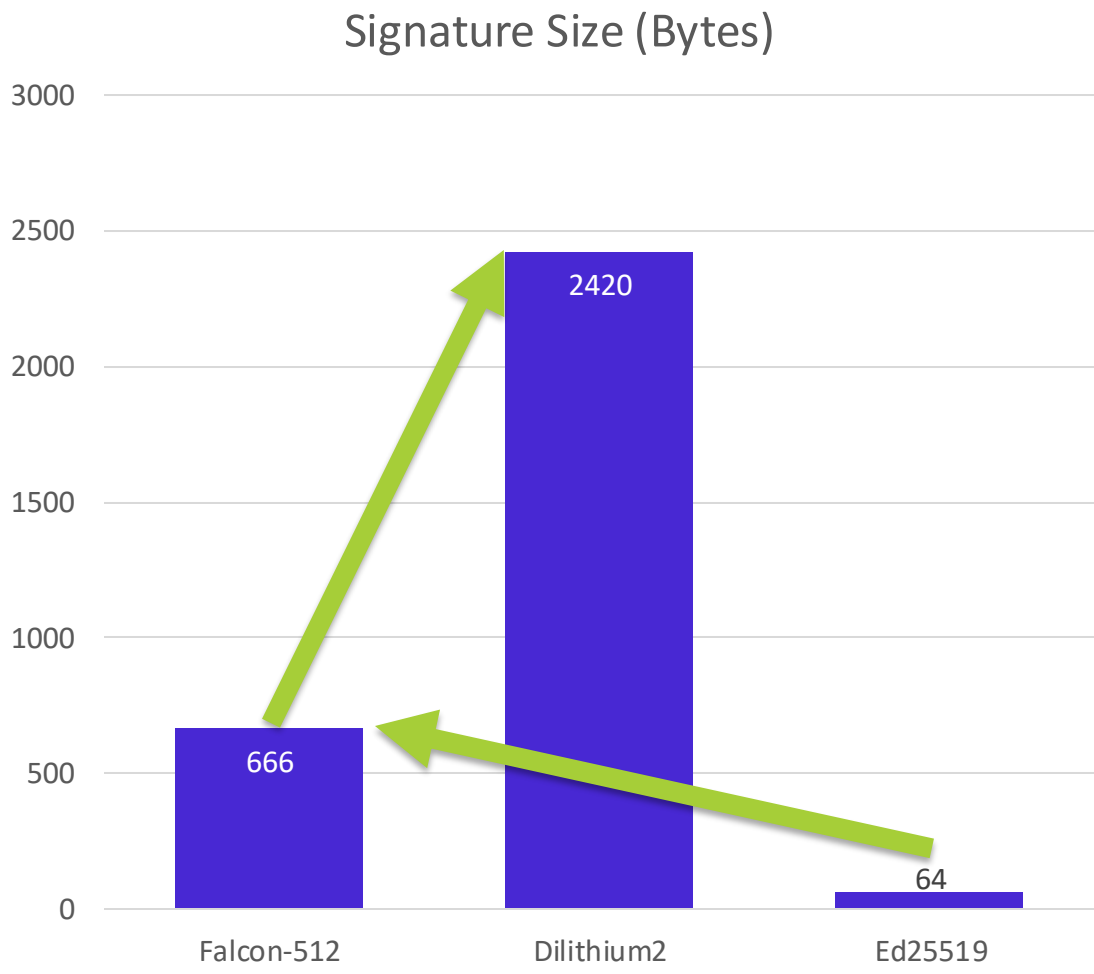
Why Falcon?

- At this point you may be wondering why not Dilithium?
  - Dilithium is much faster than Falcon!
  - There’s an order of magnitude between them!
  - Surely, it's a better choice for online/offline?



Why Falcon?

- Falcon more suited to small devices:
  - Compact Signatures 📏 – Falcon is 3.5x smaller
  - Fast Verification ⚡ – Falcon verifies 3x faster
  - Falcon is also a NIST PQC standard
- Our goal:
  - Apply Online/Offline to Falcon's slower signing times!
  - Retain Falcon's advantages
  - Make POS etc. PQ and practical!



Falcon: Under the Hood

Why is Falcon so expensive to sign?

- Many *FFT* conversions of the secret key
  - Costs more than total Dilithium2 signing
  - And this happens twice!
- *ffSampling* – trapdoor Gaussian sampler
  - It's recursive, uses a lot of randomness, and is computationally expensive
- Falcon requires *floating-point* operations
  - And on small embedded devices with no FPU means costly emulation

$\mathsf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \mathbf{B}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

Our Contributions

Sign$(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathsf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \hat{\mathbf{B}}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs more than total Dilithium2 signing
  - And this happens twice!

Sign$(sk, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \mathbf{B}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs [illegible] igning
  - And t[illegible]

Offline

$\mathsf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \vec{\mathbf{B}}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs [illegible] igning
  - And t[illegible]
- *ffSampling* – trapdoor Gaussian sampler
  - It's recursive, uses a lot of randomness, and is computationally expensive

Offline

$\mathsf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \hat{\mathbf{B}}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

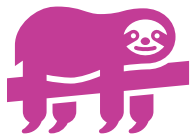
Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs … igning
  - And t…
- *ffSampling* – trapdoor Gaussian sampler
  - It's recu… ess, and is c…

Offline

Lazify



$\textbf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \mathbf{B}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

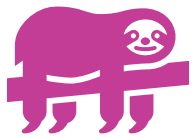
Our Contributions

- Many *FFT* conversions of the secret key
  - Costs … signing
  - And t…

Offline

- *ffSampling* – trapdoor Gaussian sampler
  - It's recu… ess, and is c…

Lazify



Falcon requires *floating-point* operations

And on small embedded devices with no FPU means costly emulation

$\textbf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \mathbf{B}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

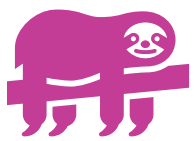
Our Contributions

- Many *FFT* conversions of the secret key
  - Costs … igning
  - And t…

Offline

- *ffSampling* – trapdoor Gaussian sampler
  - It's recu… ess, and is c…

Lazify



Falcon requires *floating-point* operations

And on small… FPU means …

$\mathsf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \mathbf{B}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

RSAC 2025 Conference

Post-Quantum Online/Offline Signatures

Lazy Falcon “Few Times” Signatures

Many Voices.
One Community.

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs more than total Dilithium2 signing
  - And this happens twice!
- *ffSampling* – trapdoor Gaussian sampler
  - It's recursive, uses a lot of randomness, expensive

$\mathsf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \mathbf{B}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

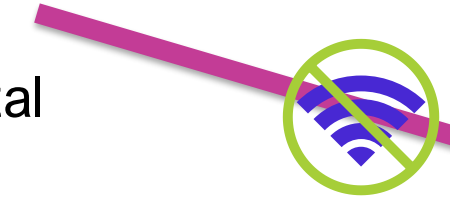
while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs more than total Dilithium2 signing
  - And this happens twice!
- *ffSampling* – trapdoor Gaussian sampler
  - It's recursive, uses a lot of randomness, expensive



PreSign(sk)

$\mathbf{B} \leftarrow [g, -f; G, -F]$

$\hat{\mathbf{B}} \leftarrow \mathsf{FFT}(\mathbf{B})$

$u_0, u_1 \leftarrow \mathcal{D}_\sigma^2$

$c_{\text{pre}} \leftarrow \mathsf{ComputeTarget}(h, u_0, u_1)$

return $(\rho := u_0, u_1, c_{\text{pre}})$

ComputeTarget(h, u_0, u_1)

$\hat{u_1} \leftarrow \mathsf{NTT}(u_1)$

$\hat{t} \leftarrow h \odot \hat{u_1}$

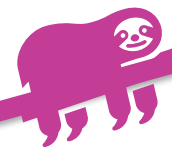
$t \leftarrow \mathsf{invNTT}(\hat{t}) + u_0$

return t

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs more than total Dilithium2 signing
  - And this happens twice!
- *ffSampling* – trapdoor Gaussian sampler
  - It's recursive, uses a lot of randomness, expensive



$\mathsf{Sign}(\mathsf{sk}, \mu, \lfloor \beta^2 \rfloor)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$\mathbf{t} \leftarrow (-\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(F), -\frac{1}{q}\mathsf{FFT}(c) \odot \mathsf{FFT}(f))$

do

  do

    $\mathbf{z} \leftarrow \mathsf{ffSampling}_n(\mathbf{t}, \mathbf{T})$;

    $\mathbf{s} = (\mathbf{t} - \mathbf{z}) \odot \mathbf{B}$;

  while $\|\mathbf{s}\|^2 > \lfloor \beta^2 \rfloor$

  $(s_0, s_1) \leftarrow \mathsf{invFFT}(\mathbf{s})$

  $s \leftarrow \mathsf{Compress}(s_1, 8 \cdot \mathsf{sbytelen} - 328)$

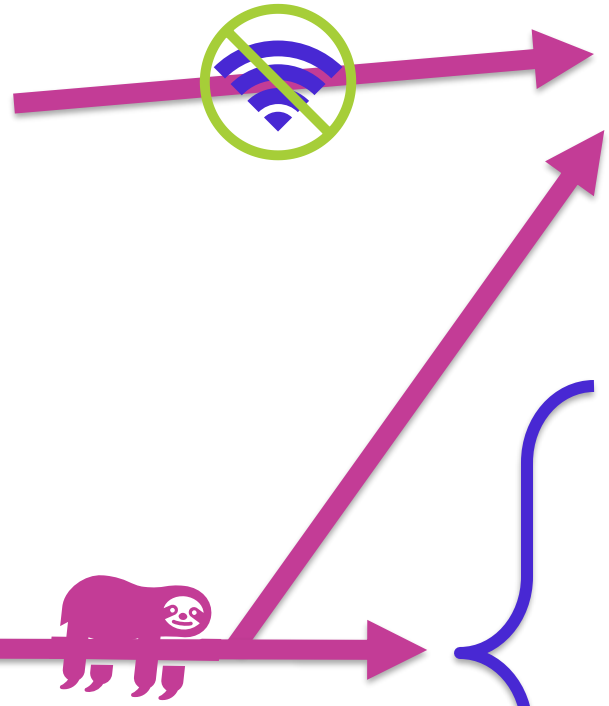
while $s = \perp$

return $\sigma := (\mathbf{r}, s)$

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs more than total Dilithium2 signing
  - And this happens twice!
- *ffSampling* – trapdoor Gaussian sampler
  - It's recursive, uses a lot of randomness, expensive



$\mathsf{Sign}(\mathsf{sk}, \rho, \mu)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c' \leftarrow \mathsf{HashToPoint}(\mathbf{r} \| \mu, q, n)$

$c \leftarrow c' + c_{\mathrm{pre}}$

$s_0', s_1' \leftarrow \mathsf{SampPre}_{\hat{\mathbf{B}}}(c, \hat{\mathbf{B}})$

$s_0 \leftarrow s_0' - u_0,\ s_1 \leftarrow s_1' - u_1$

$s \leftarrow \mathsf{Compress}(s_1)$

return $\sigma := (\mathbf{r}, s)$

$\mathsf{SampPre}_{\hat{\mathbf{B}}}(c, \hat{\mathbf{B}})$

$\hat{c} \leftarrow \mathsf{FFT}(c)$

$\mathbf{if}\ \ \hat{\mathbf{t}} = (\mathsf{FFT}(c), \mathsf{FFT}(0)) \cdot \hat{\mathbf{B}}^{-1}$

$\hat{\mathbf{t}} \leftarrow \left(-\frac{1}{q} \cdot \hat{c} \odot \mathsf{FFT}(F), -\frac{1}{q} \cdot \hat{c} \odot \mathsf{FFT}(f)\right)$

$\mathbf{t} \leftarrow \mathsf{invFFT}(\hat{\mathbf{t}});\quad \mathbf{t} \leftarrow \mathsf{round}(\mathbf{t});\quad \hat{\mathbf{t}} \leftarrow \mathsf{FFT}(\mathbf{t})$

$\hat{\mathbf{s}} \leftarrow (\mathsf{FFT}(c), \mathsf{FFT}(0)) - \hat{\mathbf{t}} \odot \hat{\mathbf{B}}$

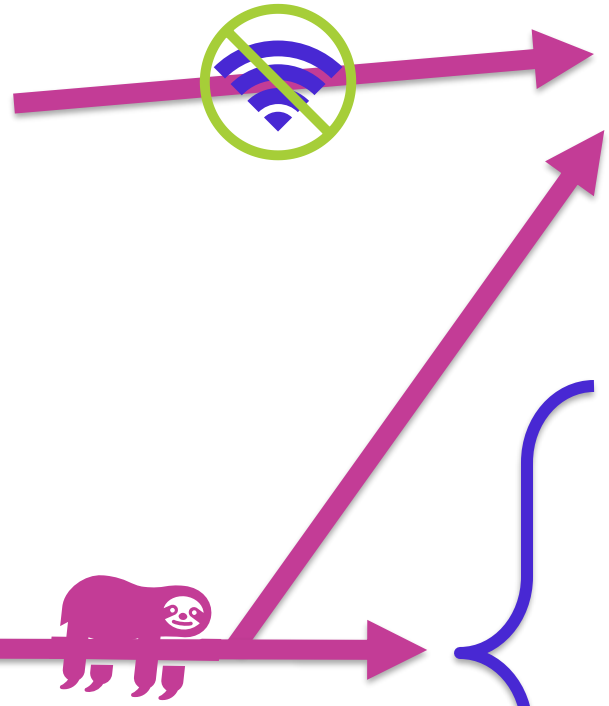
$\mathbf{s} \leftarrow \mathsf{invFFT}(\hat{\mathbf{s}})$

return $\mathbf{s}$

Lazy Falcon

Our Contributions

- Many *FFT* conversions of the secret key
  - Costs more than total Dilithium2 signing
  - And this happens twice!
- *ffSampling* – trapdoor Gaussian sampler
  - It's recursive, uses a lot of randomness, expensive



$\mathsf{Sign}(\mathsf{sk}, \rho, \mu)$

$\mathbf{r} \leftarrow\!\!\$\ \{0,1\}^{320}$

$c' \leftarrow \mathsf{HashToPoint}(\mathbf{r}||\mu, q, n)$

$c \leftarrow c' + c_{\mathrm{pre}}$

$s_0', s_1' \leftarrow \mathsf{SampPre}_{\hat{\mathbf{B}}}(c, \hat{\mathbf{B}})$

$s_0 \leftarrow s_0' - u_0,\ s_1 \leftarrow s_1' - u_1$

$s \leftarrow \mathsf{Compress}(s_1)$

return $\sigma := (\mathbf{r}, s)$

$\mathsf{SampPre}_{\hat{\mathbf{B}}}(c, \hat{\mathbf{B}})$

$\hat{c} \leftarrow \mathsf{FFT}(c)$

$\hat{\mathbf{t}} = (\mathsf{FFT}(c), \mathsf{FFT}(0)) \cdot \hat{\mathbf{B}}^{-1}$

$\hat{\mathbf{t}} \leftarrow \left(-\frac{1}{q} \cdot \hat{c} \odot \mathsf{FFT}(F), -\frac{1}{q} \cdot \hat{c} \odot \mathsf{FFT}(f)\right)$

$\mathbf{t} \leftarrow \mathsf{invFFT}(\hat{\mathbf{t}})$; $\mathbf{t} \leftarrow \mathsf{round}(\mathbf{t})$; $\hat{\mathbf{t}} \leftarrow \mathsf{FFT}(\mathbf{t})$

$\hat{\mathbf{s}} \leftarrow (\mathsf{FFT}(c), \mathsf{FFT}(0)) - \hat{\mathbf{t}} \odot \hat{\mathbf{B}}$

$\mathbf{s} \leftarrow \mathsf{invFFT}(\hat{\mathbf{s}})$

return $\mathbf{s}$

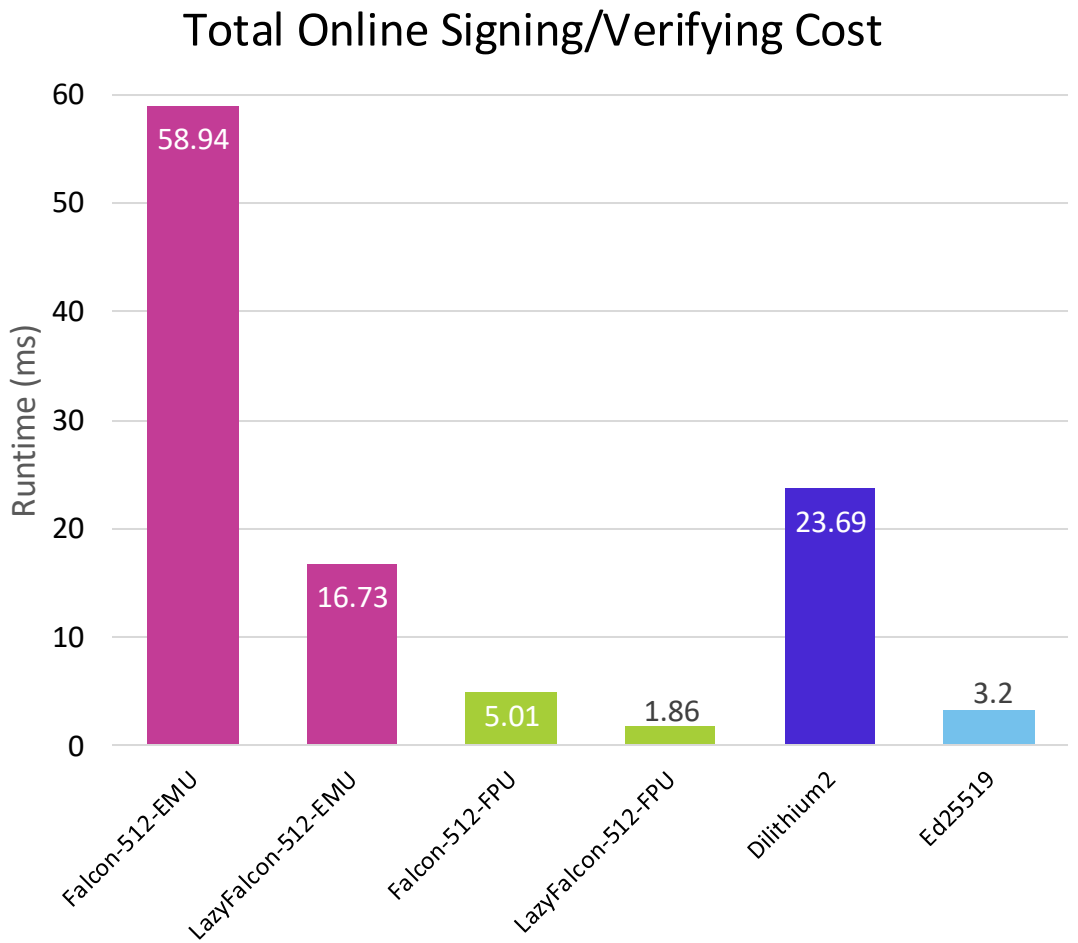
RSAC™ 2025 Conference

Benchmarking Post-Quantum Signatures

Many Voices.
One Community.

Benchmarking PQ signatures

- Pink is Falcon emulated
  - Most realistic for small/embedded
  - Shows a ~4x saving
- Green is Falcon using native FPU
  - Roughly the same savings vs Falcon
  - Nearly 2x faster than Ed25519
- Blue is Dilithium2
  - Much slower verify ~3 ms
  - Lazy Falcon at least 1.5x faster and nearly 13x faster with FPU



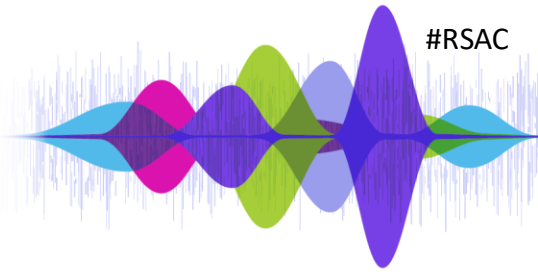
Benchmarked on Raspberry Pi 3B (ARM Cortex-A53), latest GCC

RSAC 2025 Conference

Post-Quantum Online/Offline Signatures

Lazy Falcon “Few Times” Signatures: Pros and Cons

Many Voices.
One Community.



Pros and Cons of Lazy Falcon

- Pros
  - We can sign/verify faster than Ed25519
  - Lazy Falcon is compatible with Falcon
    - Same key generation, verification has one difference
  - You *kinda* get *some* side-channel protection “for free”
- Cons
  - Sig. cost is slightly bigger, but smaller than ML-DSA/Dilithium
  - “Few-times” signature is realistically at most 4 or 5 times

RSAC 2025 Conference

Many Voices.
One Community.

Thank you

More information:

- Martin R. Albrecht, Nicolas Gama, James Howe, and Anand K. Narayanan, 2025. *Post-Quantum Online/Offline Signatures,* eprint.iacr.org/2025/117
- Reference code: https://github.com/jameshoweee/online-offline-sigs